

Willkommen

zu einer Vertiefungs-
stunde des Java-
tutoriums der FS Bioinfo

0. Einleitung

1. Java-Style schreiben
2. Schleifen
 - 2.1 for-Schleife
 - 2.2 while-Schleife
3. File Input/Output
 - 3.1 Lesen aus File
 - 3.2 Splitten
 - 3.3 Schreiben in File
4. Übungen

0. Einleitung



1. Java-„Style-Guide“

... auch genannt „Programmierrichtlinien“.

WARUM?

- Übersichtlichkeit
- „Portabilität“

... und eine Portion „Glaube“.

1. „Style-Guide“

Die (subjektiv!) 7(?) wichtigsten Regeln, um nicht negativ aufzufallen:

1. Elemente nach „Sichtbarkeit“ einrücken.
2. Klassen-, Konstanten und Konstruktornamen fangen mit Großbuchstabe an.
3. Variablen- und Methodennamen fangen klein an (und Methoden werden groß fortgesetzt z.B. **calculateMyArray()**).
4. Verwende für jede Anweisung eine neue Zeile.

„Style-Guide“ II

5. Verwende entweder deutsche oder englische Bezeichnungen aber nicht beides zusammen.
6. Keine „magic numbers“ – niemals!
7. Verwende aussagekräftige Namen für alles (außer Schleifenvariablen) – das hilft nicht nur anderen! ;)
8. Scheue dich nicht ab und zu mal eine Zeile Kommentar zu schreiben.
9. Benutze die Funktion „auto format code“ in Eclipse z.B. mit „*Strg + Shift + F*“.

„Style-Guide“: Fazit

- Ist jemandem etwas aufgefallen?

„Style-Guide“: Fazit

- Das waren keine 7 „Regeln“...

... hat das etwas zu bedeuten?

„Style-Guide“: Fazit

Gestalte deinen Code, wie er dir gefällt,
HAUPTSACHE er ist:

- 1. korrekt,**
- 2. übersichtlich und**
- 3. konsistent!**

„Style-Guide“ III

- Wer's noch genauer wissen will schaut hier:
<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>

- ...und für alle, die eine Wissenschaft draus machen wollen gibt es hier

Buchempfehlungen:

https://rz-static.uni-hohenheim.de/anw/programme/prg/java/tutorials/javainset4/javainset_26_000.htm

2. Wiederholung: Schleifen

- Das Prinzip dahinter: solange eine Bedingung (nicht) erfüllt ist, wird eine bestimmte Anweisung wiederholt. Ist die Bedingung immer erfüllt, spricht man von einer Endlosschleife.
- Die beiden wichtigsten Schleifen sind die for-Schleife und die while-Schleife.
- Theoretisch sind „for“ und „while“ gleichwertig, praktisch ist die Konvertierung nicht immer schön und/oder einfach.

2.1 Schleifen II: for-Schleife

- Syntax:

for (Initialisierung; Schleifenbedingung; De-/Inkrement) {Anweisung;}

- Zum Beispiel so:

```
for (int i = 0; i < 10; i++){  
    system.out.println  
        („An Pos „+ i +“ steht: „ + anyArray[i]);  
}
```

2.2 Schleifen: while-Schleife

- Syntax: *while (Bedingung) {
Anweisung; }*

- Könnte so aussehen:

```
int i = 0;  
while (i < anyArray.length){ System.out.println (someArray[i]);  
i++;  
}
```

2.3 Schleifen: Was tun?

- „for“-Schleife verwenden bei „bekannter“ Anzahl von Durchläufen
- „while“-Schleife bei unbekannter Anzahl
- Anmerkung: von dem „break;“-Kommando sollte man nur in Ausnahmefällen Gebrauch machen. Die Schleifenvariablen der for-Schleife während des Ablaufs manuell zu verändern ist ebenfalls sehr schlechter Stil.

2.4 Übung nötig...

...oder gewünscht?

2.4 Übung:

- Collatz-Problem (siehe auch Ulam-Zahlen)
 - <http://de.wikipedia.org/wiki/Collatz-Problem>
1. Wähle eine natürliche Zahl
 2. Ist n gerade, so nimm als nächstes $n / 2$
 3. Ist n ungerade, so nimm als nächstes $3n + 1$
- Die Collatz-Vermutung lautet:
Jede so konstruierte Zahlenfolge endet im Zykel 4,2,1 egal, mit welcher natürlichen Zahl man beginnt.

2.4 Übung II

```
public class Collatz {  
  
    public static int input = 10;  
  
    public static void main(String[] args) {  
  
        System.out.println("Checke Collatz-Vermutung für " + input + ": ");  
  
        while (input != 2) {  
            if (input % 2 == 0) {  
                input = input / 2;  
            } else {  
                input = 3 * input + 1;  
            }  
            System.out.print(input + ", ");  
        }  
        System.out.println("1 => true!");  
    }  
}
```

2.4 Übung III

- Wo findet die „for“-Schleife Verwendung?
- Berechne die Collatz-Vermutung für eine Reihe von Zahlen, z. B. die Reihe von Input hinunter bis zur 2.
- Alternativ: 10 Vielfache des Inputs

3. File I/O

- Bisher können wir nur Programme schreiben, die bei jedem Durchlauf, das gleiche Ergebnis berechnen und nach dem beenden alles „vergessen“.
- Input von Dateien oder Benutzereingaben verarbeiten
- Ergebnisse dauerhaft speichern (Output)

3.1 File Input:

- Lesezugriff auf eine Datei, für gewöhnlich Zeile für Zeile
- Mit IDE werden die nötigen Ressourcen und Exceptions semi-automatisch geladen.
- Weitere Details siehe Folien von 17.11

3.1 File Input II

```
FileReader fr = new FileReader (myfile);  
BufferedReader br = new BufferedReader(fr);
```

```
String s = null;  
while ((s = br.readLine()) != null) {
```

```
    VERARBEITUNG von s;
```

```
}  
fr.close();  
br.close();
```

3.1 File Input III

- Wichtiges Kommando für Verarbeitung:
String[] lineSplit = s.split("\\s+");
Trennt nach Leerzeichen auf.
- „Suche“ nach bestimmten Begriffen mit
lineSplit[0].equals(„KEYWORD“);

3.2 File Output

- Analog zu unserem Input der Schreibzugriff:
- Wirft ebenfalls eine Exception, wenn z.B. kein Schreibrecht, Platte voll, etc.

3.2 File Output II

```
FileWriter writer;  
File file = new File ([<Pfad>] <Dateiname>);  
writer = new FileWriter (file, true*);  
writer.write(„myText“ + myInteger);  
writer.write(System.getProperty("line.separator"));  
writer.flush();  
writer.close();
```

- * true schreibt in Datei, falls vorhanden; false löscht vorhandene Dateien gleichen Namens automatisch!

3.3 Übung:

1. Schreibe die Ausgabe eines Programms (z.B. Collatz) in ein Textfile.

DANKE...



...für die Aufmerksamkeit, die weitere Zeit steht für Übungen und Fragen zur Verfügung.